International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

Study of Process Models and Certification of Components

Kiran Bala[1], Vijay[2]

[1, 2]Department of Computer Science & Engineering, Meri College of Engineering & Technology, Sampla, Bhadurgarh, Haryana-(India)

Abstract: ***Component-Based Software Engineering is the process of building software systems from reusable parts which offers the potential to radically advance the way in which software is developed. Component Based Software Development (CBSD) is focused on assembling existing components to build a software system, with a potential benefit of delivering quality systems by using quality components. It departs from the conventional software development process in that it is integration centric as opposed to development centric. The quality of a component-based system using high quality components does not therefore necessarily guarantee a system of high quality, but depends on the quality of its components, and a framework and integration process used. It leads to increased reuse leading to higher quality and reduced time to market. "Higher quality" means that the components must have a quality stamp in terms of what level of reliability can be expected from them. Thus, the certification stands out as an essential area to evaluate the component reliability level. This paper presents a survey of software component certification research.***

Keywords: Challenges, CBS, Component types, COSE, CSLC, High Quality, Prototyping.

1. Introduction

Component-based software development has emerged as a viable and economic alternative to the traditional software development process. The ability to build complete system solutions by interconnecting components through public interfaces independently created and deployed components is the driving force behind Component- Based Software Engineering (CBSE). However, organizations reusing existing software components can only achieve the improvements related to software reuse if the selected software components have a certain quality degree. A major problem when building software systems from components is the unknown quality of the components and the unknown side-effects of their integration. In our view, component evaluation has to be performed at different stages in the component life cycle, by different actors, for different reasons. During component development, the component vendor assures the quality of the components developed before made publicly available for reuse. Component certification means that an independent actor performs an evaluation according to standardized procedures, so that an issued certificate is seen as a quality stamp which increases the trust in the component. Reuse is a "generic" denomination encompassing a variety of techniques aimed at getting the most from design and implementation work. The top objective is to avoid reinvention, redesign and reimplementation when building a new product, capitalizing on previously done work that can be immediately deployed in new contexts. Therefore, better products can be delivered in shorter times, maintenance costs are reduced because an improvement to one piece of design work will enhance all the projects in which it is used, and quality should improve because reused components have been well tested [2].

2. Objective of the Study

Certification of components is the challenge of Component-Based Software Engineering. This study especially aims to reveal certification of Components and components based systems. As components are available from different sources their reliability and Quality Assurance is required. Various testing techniques are there to test the components but there must be some standards and model by which we can assure about the quality of components and systems made from these components. I in this study tried to explain the certification of components and component-based systems that holds the components from various parties.

3. Various Process Models

A software system, either small or large scale, is an uncertain concept at the beginning and therefore needs to be analyzed, designed and implemented. This completes the development but it is not over in terms of operation of software. Maintenance is required after all steps to keep software alive. All of these steps are called software process model.

a) Traditional Process Models

Since each product shows different characteristics at development stage, various project life cycles can be applied to computerized systems development. Some of them will spend years in the logical phase, current hardware may just not be fast enough for the product, or current users may not be capable of this new system and computerized background. Some of them can also be quickly designed and implemented and then many years are spent for modifications to meet the users' changing needs in the maintenance phase.

A number of different life-cycle models will be described and three of them are most widely used:

- Waterfall with iteration,
- Rapid prototyping and
- The spiral model, which received considerable attention recently.

b) Rapid Prototyping Model

Since users operate the system and perform fixed processes, they cannot imagine the whole system most of the time and they need to be conveyed into a working system. Therefore,

it is a problem not to define the requirements entirely in terms of development processing. As well as being used to investigate the requirements, prototyping might also be used to discover the most suitable form of user interfaces.

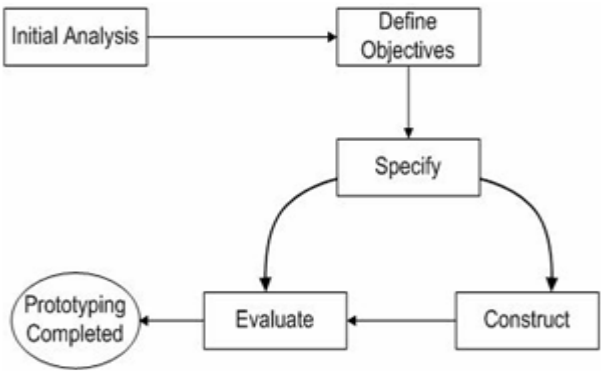

Figure 1: Rapid Prototyping Model

Some advantages and disadvantages of Rapid Prototyping Model [1] are described below

Advantages of Rapid Prototyping Model

1) Early demonstrations of system functionality help identify any misunderstandings between developer and client;
2) Client requirements that have been missed are identified;
3) Difficulties in the user interface can be identified;
4) The feasibility and usefulness of the system can be tested, even though, by its very nature, the prototype is incomplete.

Disadvantages of Rapid Prototyping Model

1) The client may perceive the prototype as part of the final system, or may not understand the effort that will be required to produce a working production system, and also may expect delivery soon.
2) The prototype may divert attention from functional to solely interface issues.
3) Prototyping requires significant user involvement.
4) Managing the prototyping life cycle requires careful decision-making.

A solution is offered that combining the two approaches, waterfall and prototyping.

4. Incremental Model

In the incremental model, operational products are delivered at each stage. Software is not written; it is built [4]. The complete product is divided into builds, and the developer delivers the product build by build. A typical product consists of 10 to 50 builds. At each stage, the client has an operational quality product that does a portion of what is required; from delivery of the first build, the client is able to do useful work.

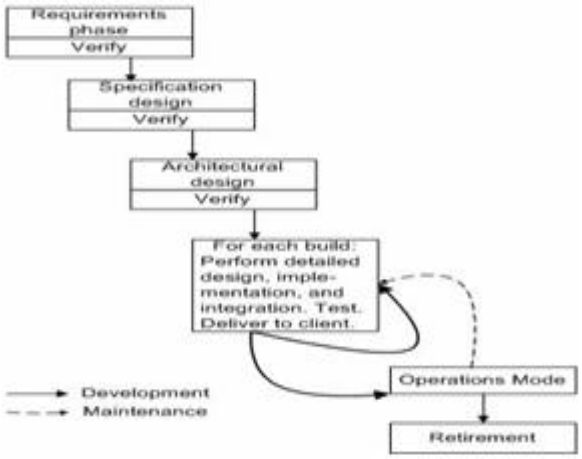

Figure 2: Incremental Model

With the incremental model [4], portions of the total product might be available within weeks, whereas the client generally waits months or years to receive a product built using the waterfall or rapid prototyping models. Another advantage of the incremental model is that it reduces the traumatic effect of imposing a completely new product on the client organization. From the client's financial viewpoint, phased delivery does not require a large capital

Component Baesed Software Engineering Process Models

The component-based software life cycle (CSLC) is the life cycle process for a software component with an emphasis on business rules, business process modeling, design, construction, continuous testing, deployment, evolution, and subsequent reuse and maintenance. In general, analysis and design phases for component-based process models take more time than traditional ones take.

5. Stojanovic Process Model

A component-oriented development process model, is shown in Figure 3

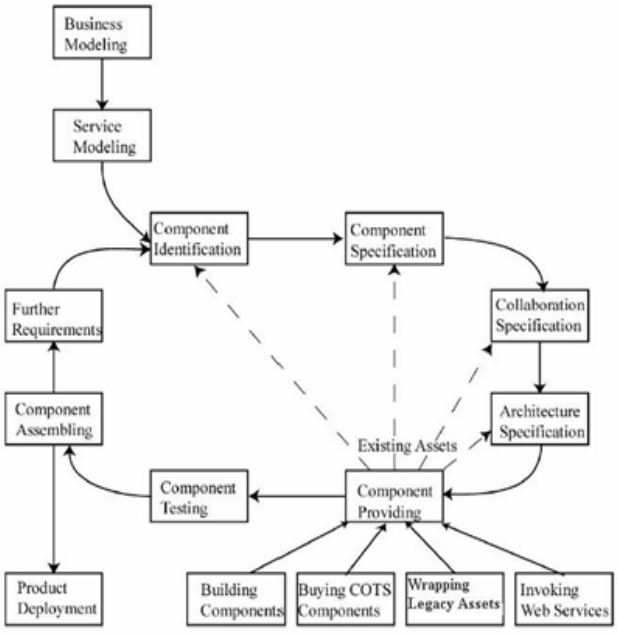

Figure 3: Stojanovic Process Model

It has been introduced by Stojanovic [5], focusing on the component concept from business requirements to implementation. This process model is called by its owner's name. The phases of requirements, analysis, design and implementation in a traditional development process has been substituted by service requirements, component identification, component specification, component assembly and deployment. After the components of the system are fully specified, a decision can be made to build components, wrap existing assets, buy COTS (Commercial Off-the- Shelf) components or invoke web services over the Internet.

6. Component Oriented Software Engineering Process Model

Figure 4 illustrates steps of COSE Process Model [3] in general, which is summarized here to reveal COSE Process Model and its details; COSE Process Model building activity starts top-down to introduce the building blocks of the system. As the activity continues towards lower granularity blocks, interfaces between the blocks are also defined. At an arrived level where the module is expected to correspond to a component, a temporary bottom-up approach can be taken; if desired capability can only be achieved by a set of components, their integration into a super-component should be carried out.

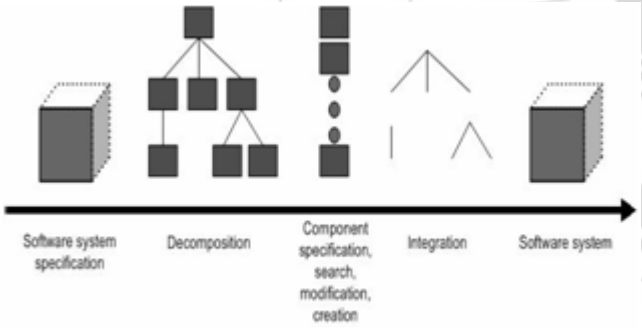

Figure 4: Component Oriented Software Engineering (COSE) Process Model

7. Component

The whole comprises its parts, and the parts compose the whole. To compose, from the Latin com- "together" and ponere "to put." The parts, which are composed, are etymologically components. The phrase component-based system has about as much inherent meaning as "part-based whole" [6].

1. According to **Brown and Wallnau** "A component is a non-trivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces."
2. According to **Councill and Heinmann** "A software Component is a software element that confirms to a component model and can be independently deployed and composed without modification according to a composition standard."

Types of Components

In addition to COTS components & Open source components, the CBSE process [1] yields:

1. Qualified components
2. Adapted components
3. Assembled components
4. Updated components

Qualified components—assessed by software engineers to ensure that not only functionality, but also performance, reliability, usability, and other quality factors conform to the requirements of the system or product to be built. [1]

Adapted components—adapted to modify (also called mask or wrap) unwanted or undesirable characteristics [1].

Assembled components—integrated into an architectural style and interconnected with an appropriate infrastructure that allows the components to be coordinated and managed effectively [1].

Updated components—replacing existing software as new versions of components become available [1].

8. Certification

"To attest as certain; give reliable information of; confirm, to testify to or vouch for in writing, to guarantee; endorse reliably; to certify a document with an official seal."

- [MWU 96]

Need of Certification

In the present business scenario every developer and company wants to produce components in fast and cost effective manner by using ready-made components if available. This process requires buy or make decision to follow. If we buy it from third party then its quality must be as required as well as it is better if it is certified from some vendor or agency for compliance with the standard and Quality model [7]. There are various models to evaluate the quality from existing software quality models [7], namely McCall's, Boehm, ISO 9126, FURPS, Dromey, ISO/IEC TR 15504-2 1998(E), Triangle and Quality Cube.

9. Challenges in Maintaining Certification

Introducing a component can affect the safety reasoning of the system [8]. This becomes more apparent in the case of an existing system undergoing upgrades (as in the case of the Hawk). Introducing a new component can affect the safety reasoning of the system in the following ways:

1) **New dependencies and services:** The operation of new modules (provided by the component) that did not exist in the previous version of the system may require different behavior (functionality) from the system modules that were not changed. This can include functionality that is different but within the capabilities of the existing modules. Although the new modules are not affected the way they interact with the components is altered. Hence the safety properties of this new interaction need to be examined and captured in safety case. Furthermore, the

component may provide slightly different behavior than the bespoke modules that it replaces (e.g. slightly different properties or memory management). The new services the components offers will also need to be analyzed.

2) **Change of context:** During the evolution of the safety case, the safety argument is based on a number of (documented) contextual information such as operational information, development information and assumptions. A change may challenge the contextual information associated with the argument. For example, the development of a COTS component may be compliant with a different standard than the one documented in the safety case.
3) **New or updated evidence:** An argument about the safe operation of a system (or parts of the system) is substantiated by evidence. A safety case can include different types of evidence, such as qualitative evidence, expert opinion and direct evidence produced during testing. A new component will bring in the safety case a new set of evidence, whereas a modified component would need to have its associated evidence updated to cover the change.

10. Conclusion

This paper has presented a survey related to the certification of components and component based system in the software component certification research. This paper defines the components and component types.Also this paper describes the certification and need of certification followed by challenges in maintaining certification. Through this survey, it can be noticed that software components certification is still immature and further research is needed in order to develop processes,
methods, techniques, and tools aiming to obtain welldefined standards for certification.

References

[1] Pressman Roger S, Software Engineering, Tata McGraw Hill, 2006
[2] Herzum Peter, Sims Oliver, Business Component Factory, Wiley, 1999
[3] Dogru Ali H., Tanik Murat M., A Process Model for Component- Oriented Software Engineering, IEEE Software, March/April 2003
[4] Schach Stephen R., Classical and Object Oriented Software Engineering, 4th Edition, McGraw Hill, 1999
[5] Stojanovic Zoran, An Integrated Component-Oriented Framework for Effective and Flexible Enterprise Distributed Systems Development, http://www.betade.tudelft.nl/publications/Stojanovic_CAISE2002.pdf
[6] Khalid Eldrandaly, The International Arab Journal of Information Technology, Vol. 5, No. 3, July 2008, Pg 304-311.
[7] Rawashdeh A and Matalkah B, Journal of computer Science 2, 2006, ISSN 1549-3636 2006 Science Publications, Pg 373-381
[8] Cai Xia, LYU Michael R., wong Kam-Fai, KO Roy, Component-Based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes